**Lab 9:**

Hassan Shahzad – 211798

**Task 1:**

#include <stdio.h>

#include <pthread.h>

#include <time.h>

#define MAX 16

#define NTHREADS 4

void \*thread\_function(void \*);

pthread\_mutex\_t mutex1 = PTHREAD\_MUTEX\_INITIALIZER;

int total\_sum = 0;

int a[] = { 1, 5, 7, 10, 12, 14, 15, 18, 20, 22, 25, 27, 30, 64, 110, 220 };

int part = 0;

int main()

{

clock\_t start\_t, end\_t, total\_t;

pthread\_t thread\_id[NTHREADS];

int i, j;

start\_t = clock();

for(i=0; i < NTHREADS; i++)

{

pthread\_create( &thread\_id[i], NULL, thread\_function, NULL );

}

for(j=0; j < NTHREADS; j++)

{

pthread\_join( thread\_id[j], NULL);

}

/\* Now that all threads are complete I can print the final result. \*/

/\* Without the join I could be printing a value before all the threads \*/

/\* have been completed. \*/

printf("Final counter value: %d\n", total\_sum);

end\_t = clock();

total\_t = (double)(end\_t - start\_t);

printf("Total time taken by CPU: %ld\n", total\_t );

return 0;

}

void \*thread\_function(void \*dummyPtr)

{

//printf("Thread number %ld\n", pthread\_self());

int thread\_part = part++;

for (int i = thread\_part \* (MAX / 4); i < (thread\_part + 1) \* (MAX / 4); i++){

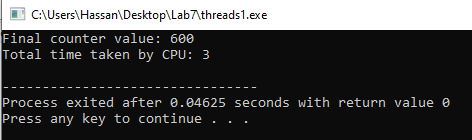
pthread\_mutex\_lock(&mutex1);

total\_sum += a[i];

pthread\_mutex\_unlock( &mutex1 );

}

}



**Task 2:**

#include <stdio.h>

#include <pthread.h>

#include <time.h>

#define MAX 16

#define NTHREADS 4

void \*thread\_function(void \*);

int total\_sum = 0;

int a[] = { 1, 5, 7, 10, 12, 14, 15, 18, 20, 22, 25, 27, 30, 64, 110, 220 };

int part = 0;

int sum[4] = { 0 };

int main()

{

clock\_t start\_t, end\_t, total\_t;

pthread\_t thread\_id[NTHREADS];

int i, j;

start\_t = clock();

for(i=0; i < NTHREADS; i++)

{

pthread\_create( &thread\_id[i], NULL, thread\_function, NULL );

}

for(j=0; j < NTHREADS; j++)

{

pthread\_join( thread\_id[j], NULL);

}

/\* Now that all threads are complete I can print the final result. \*/

/\* Without the join I could be printing a value before all the threads \*/

/\* have been completed. \*/

for(int abc= 0;abc<NTHREADS ; abc++){

total\_sum +=sum[abc];

}

end\_t = clock();

total\_t = (double)(end\_t - start\_t);

printf("Final counter value: %d\n", total\_sum);

printf("Total time taken by CPU: %ld\n", total\_t );

return 0;

}

void \*thread\_function(void \*dummyPtr)

{

//printf("Thread number %ld\n", pthread\_self());

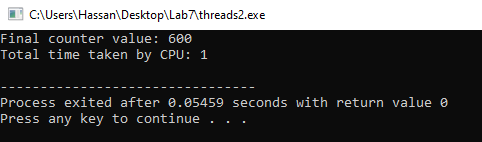
int thread\_part = part++;

for (int i = thread\_part \* (MAX / 4); i < (thread\_part + 1) \* (MAX / 4); i++){

sum[thread\_part] += a[i];

}

}



**Task 3:**

Which approach do you think would run faster, and why?

**Answer**:

The clock time also shows the time for the task to run. Task 2 takes less time than task 1 as in task 2, as threads are working concurrently whereas, in task 1 threads execute one by one. Plus in task 1, the threads excess a single global variable whereas, in task2 they access different sum array.